

6700-722

Specification of the Unified User-Level Protocol

This document represents the results of various inputs, both in response to RFC 451 and from other sources. The dominant other source was my friendly neighborhood Principal Investigator, who told me that the ARPA office is quite interested in the specification of a Network-wide user interface. As just such a thing was the underlying motivation of the tentative proposal for a unified user-level protocol (UULP), it seemed appropriate to come up with the following. It is being distributed in draft form to the attendees of the Resource Sharing Workshop, with the expressed understanding that many details are subject to change.

Context

The UULP is initially to be a negotiated option, in the sense of the Telnet Protocol. When the option is enabled, the Server Host will furnish a command environment which supports the conventions and commands discussed herein. Any optional commands which are not supported by a particular Server are to be responded to by the following message: "Not implemented: commandname.", where commandname is the name of the command which was requested. Note that throughout this document, all literals must be sent exactly as specified, so as to allow for the possibility of Servers' being driven by users' programs ("automata", or "command macros") in addition to "live" users.

The intent of the protocol is to furnish a "common command subset" which will allow users to use various Servers over the Network without being constrained to learn the various "command languages" which abound. That is, in the spirit of the Telnet Protocol's "common intermediate representation" approach to solving the local-conventions problem, the UULP takes the view that it is far better for the user to be able to say "who" on any Host in order to learn who else is logged in than to have to remember whether on this particular Host it's "sysstat", "s.wnotc", "listf tty", or even "who". In other words, functions which are in some sense generic among the Servers are to be invoked in a common fashion. Where the command aspects of other user-level protocols are clearly generic, they too are to be invoked via the UULP. Note that only minimal assumptions are made about a Server's implementation. In general, the goal is to allow the UULP to "front end" onto local implementations.

To exit from the UULP environment, the UULP command is "local" (abbreviation: "loc"; see also the discussion of Case, below). Note that all commands terminate in Telnet "Newline" (currently cr-lf), unless altered by the "eol" command (below); internal separator is space (blank). There are two reasons for introducing a mechanism other than the apparently natural one of simply re-negotiating the option: First, it is found to be more

convenient for the user to type a command than to escape to his User Telnet program to cause the option disabling. Second, it is hoped that eventually the UULP will be legislated to be the default environment encountered by any Network login, in which case the natural way to enter the Server's "native" command environment would be by UULP command.

Note: all UULP commands discussed herein are listed in Appendix 1, categorized as to optionality, with abbreviations and brief descriptions shown. The appendix may be taken as a first-pass UULP Users' Manual.

Case

The wishes of the Telnet Protocol to the contrary notwithstanding, the question of what to do about users at upper-case-only terminals has long been a thorny one. Some Servers have no problems in such circumstances, as they operate internally in all upper-case or all lower-case and merely map all input appropriately. Other Servers, however, attach the normal linguistic significance to case. (E.g., Smith's name is "Smith" -- not "SMITH", and not "smith".) To minimize superfluous processing for those Servers which are indifferent to case, all UULP commands are to be recognized as such whether they arrive as all upper-case or all lower-case. (They will be shown here as all lower merely for typing convenience.) Note that arbitrarily mixed case is not recognized, as it is an unwarranted assumption about local implementation to suppose that input will necessarily be case-mapped.

Any Server which does distinguish between upper- and lower-case in commands' arguments (a.k.a. parameters) must furnish a UULP "map" command as specified in Appendix 2 in order to support logins from upper-case-only terminals attached to User Hosts which either do not support the Telnet Protocol's dictum that all 128 ASCII codes must be generable, or support it awkwardly. This seems a simpler and preferable solution than the alternative of legislating that upper-case Network-wide personal identifiers (and perhaps even Network Virtual Path Names) be pre-conditions to a usable common command subset. (As noted below, these latter concepts will fit in smoothly when they are agreed upon. The point here, though, is that we need not deprive ourselves of the benefits of a UULP until they are agreed upon.) The "map" command should not be a problem, for the only Host which is known to be a respecter of cases already has one.

User Names

As implied above, the various Servers have their various ways of expressing users' names. Clearly, the principle of economy of memory dictates that there should be a common intermediate representation of names in and for the Network. It is probably also clear that this representation will be based upon the

Network Information Center's "NIC ID's". However, it is unfortunately simply clear that an acceptable mechanism for securing up-to-date information cannot be legislated here -- muchless a mechanism for securely updating the implied data base. Therefore, at this stage it seems to be the sensible thing to specify only the UULP syntax for conveying to the Server the fact that it is to treat a user name as a Network-wide name rather than as a local name, and let the supporting mechanisms evolve as they may.

The prefacing of a name with an asterisk ("*") denotes a Network-wide name. (Such names may be either all upper-case or all lower-case, as with UULP commands' names.) The name "*free" is explicitly reserved to mean that (in the context of logging in) a login is desired on a supported or sampling account, if such an account is available. The response if no such account is available is to be "Invalid ident: *free." When Network-wide names are generally available Servers will either map them into local names or cause them to be registered as local names as they prefer. The point is that a Network-wide name will be "made to work" by the Server in the context of the UULP.

Special Characters and Signals

Another area in which the facts of life must outweigh the letter of the Telnet Protocol is the user's convenience is to be served is that of "erase" and "kill" characters. It is possible that User Telnets will uniformly facilitate the transmission of the Telnet control codes for generic character erase and generic line kill. It is certain, however, that User Telnets will differ -- and users will, if they use more than one User Telnet, be again placed in the uncomfortable position of having to develop too many sets of reflexes. Therefore, the UULP will optionally support the following commands: "erase char" (abbreviation: "er") and "kill char", where char is a printable ASCII character (to avoid possible conflicts with "control characters" which are recognized in the innermost areas of particular operating systems). These commands are supplements to the related Telnet control codes, and have the same meanings. The point here is that it may be far more convenient for a user to be able to say "erase #" and get the "#" to be recognized as the erase character by the Server than for the user to get his User Telnet to send the Telnet equivalent. The commands are designated as optional because they may lead to severe implementation problems on some Servers, and because the equivalent functions do, after all, exist in Telnet.

Note: the erasing is assumed to be performed "as early as possible". That is, the sequence "erase x" "erase x" should come out equivalent to "erase x" "erase" -- the second invocation resulting in the erasing of the space in the command line. Presumably, this is a sufficiently uncommon path that anomalous results would be tolerated by the user

community, but the intent ought to be clear.

The Telnet "synch" and "break" mechanisms are, by their very nature, best left to Telnet. End of line, however, might well be a different story. Therefore, as a potential convenience, the UULP optionally supports "eol char" to ask the Server to treat char as the end of line character thenceforth. To revert to Telnet newline, "eol" (i.e., no argument, current terminator).

Prompt

Another aspect in which Servers vary while being the same is how they indicate "being at command level". Some output "ready messages"; others, "prompt characters". For the UULP, where some functions will be performed by means of a command's logging in to another system, the ability to specify a known prompt character is extremely desirable. The UULP command is "prompt char" (abbreviation: "pr"), where char is the character which is to be sent when the user's process (on the Server) is at command level. It is explicitly permitted to prefix char to a line consisting of a "native" prompt or ready message. Also, this command is explicitly acknowledged to be permissible prior to login.

Note: "prompt", "eol", "erase", and "kill" may all be re-invoked with a new value of char in order to change the relevant setting; all may be turned off by invocation with no argument.

Login

Perhaps the stickiest wicket of them all is the attempt to specify a generic login, but here we go. The UULP login command is "login userid" (abbreviation: "in"), where userid is either a locally-acceptable user identifier or a Network-wide identifier as discussed above. Note that for utility in contexts to be discussed later, the locally-acceptable form must not contain spaces. (Again, the only Host known to have problems on this score is prepared to cope -- with a period (".") between personal name and project name.) Servers may respond to the login attempt with arbitrary text (such as a "message of the day"), but some line of the response must be one of the following: a prompt (as discussed above; indicating, in the present context, successful login); "Password:"; or "Invalid ident: userid." When passwords are required, it is the Server's responsibility either to send a mask or to successfully negotiate the Hide Your Input option.

Note that "login *free" is specifically defined to require no password. (If a "freeloader" has access to a User Telnet and has learned of the "*free" syntax, it is fruitless to assume that he couldn't have also read the common password.) If a password must be given, acceptable responses are arbitrary text containing a line beginning either with a prompt or with "Login unsuccessful."

or with "Account:". In the latter case, responses to the account must be one of the former two. If any errors occur during the login sequence, users are to re-try by starting from the login command. (I.e., it is not required that the Server "remember" idents or passwords.)

It is explicitly acknowledged that an acceptable response to "login *free" is "limited access only." (followed by a prompt). This is intended to warn (human) users that the free account on the Server in question exists only to allow such functions as accepting mail and telling if a particular user happens to be logged in. (For objections to "loginless" performance of such tasks, see RFC 491. Note also that nothing here says that a Server must do anything other than return a prompt in response to "login *free" in the event that loginless operation is natural to it.) Given the UULP login discipline and the "prompt" command, it is reasonably straightforward for a program to login on a free account and perform one of these functions, for if the login command succeeded, the program will "see" a guaranteed prompt character.

To make life simpler for those Hosts which normally have some sort of "daemon" process service mail and the like, a further expansion to login is in order. The point here is that some Hosts may not know what sort of process to pass an unqualified "login *free" to, whereas they'd be sure what to do with an explicit request to process mail, do a who command, or set up console to console communications. Therefore, UULP "login" will allow a "control argument" (see below) of either "-mail", "-who", or "-concom", and the respective UULP commands involved must use the respective strings in any login line they transmit. Again, nothing is being said about what a Server has to do with the information, but some Servers need/want it.

Usage Information

Most Servers offer some sort of on-line documentation, from calling sequences of commands to entire users' manuals. There are two sorts of information of interest in the UULP environment: "normal" system information, and information about the particular Server's UULP implementation. To learn how to get descriptions of "native" commands, the UULP command is "help -sys" (abbreviation: "?"). Note that "-sys" is viewed as a "control argument" and as such prefaced by a hyphen ("-") to facilitate distinction from other sorts of name (e.g., command names). To get a description of the Server's UULP implementation, "help -uulp". To get a description of a particular UULP command's implementation, "help command". To be reminded of how to use the help command, "help".

Note: as with command names and Network-wide user names, control arguments may be either all upper-case or all lower-case.

It is specifically acknowledged that "No peculiarities." is an appropriate response to "help command" if nothing of interest need be said about the Server's implementation of the UULP command in question. (After all, we're sparing users the necessity of studying a dozen or so users' manuals; the least they can do is to read the UULP command list.) Appropriate information for less taciturn Hosts to furnish would be such data as local command invoked (if such be the case), argument syntax (e.g., pathname description, or name of help file about pathnames), "To be implemented.", or even "Not to be implemented."

"Mail"

Even though a separate mail protocol is being evolved for general purposes, the UULP needs to address this topic as, by virtue of being login based, it allows systems which do access control and sender authentication on mail to make these abilities available to users within its framework of generic functions. Therefore, to read one's mailbox, the UULP command is "readmail" (abbreviation: "rm"). To have "live" input collected and sent to a local user, "mail userid"; to a remote user, "mail userid -at hostname", where the arguments have the "obvious" meanings. To send a previously-created file, "mail -f filename userid -at hostname". Several useridents may be furnished; the delimiter is space (blank). Similar considerations apply to hostnames. If both are lists, they could be treated pairwise. (A more elaborate syntax could be invented to deal with the desire to send to several users at a given host and then to other users at other hosts, but it seems unnecessary to do so at this point, for multiple invocations would get the job done.)

The mail command prefaces the message with a line identifying the sender (Host and time desirable, but not mandatory). For "live" collection, the end of message is indicated by a line consisting of only a period (".") followed by the regnant line terminator (usually the Telnet Newline, but see also the discussion of the eol command). If remote mail is not successfully transmitted, it is to be saved in a local file and that file's name is to be output as part of the failure message. ("Queueing" for later transmission is admired, but not required.) The transmission mechanism will follow the general mail protocol. Note that when invoked with a "-at" clause, the mail command will send "login *free-mail" to the remote Host(s), followed by a mail command with no "-at" clause.

A desirable, but not required, embellishment to "readmail" would be the accepting of a Host name ("-at hostname") to cause the local Host to go off to the named Host (via "login *free-mail") and check for mail there. Several hostnames could, of course, be specified. A further embellishment, which would probably be quite expensive, would be to accept "-all" as a request to check all Hosts (or, perhaps, all Hosts known to have a free account

for the purpose) for mail.

Direct Communication

The ability to exchange messages directly with other logged in users is apparently greatly prized by many users. Therefore, despite the fact that there is a sense in which this function is not within the purview of the UULP, we will address it, after a digression.

Digression: The UULP assumes that there can be straightforward "front ends" at the various Servers which translate generic function calls in a common spelling to calls for specific, pre-existing "native" functions. In the area of console to console communications, however, this premise does not really hold. The problem is that both major "native" implementations known to the author are seriously flawed. The TENEX "link" mechanism is both insecure (you've got no business seeing everything I type even if I'm careless enough to let you) and inconvenient (why should I be forced to remember that pesky semi-colon? how do I get back into phase after I've forgotten one?). It is also likely to be extremely difficult to simulate on systems which do not force Network I/O through local TTY buffers, even if the user interface were not subject to criticism. The Multics "send_message" mechanism, on the other hand, has a more sophisticated design, but is absurdly expensive. Therefore, the UULP mechanism to be described assumes that, for this function, new local implementations will be developed to support it.

The prerequisites for establishing communications are to find out if the user is logged in, and what "address" to use if so. The mechanism for gathering this information is an expanded "who" command. (Note that "who" is the UULP command to invoke the generic who's logged in function, with no constraints on format of reply.) The syntax is "who userid -at hostname", where both arguments may be multiple. If no "-at" clause, then check local Host only. Response must begin "From hostname: userid:" followed by either an appropriate address (e.g., "11" if local "concom" (see below) uses TTY numbers and userid is logged in on TTY 11), or "Not logged in."

As with mail, a "-all" embellishment might be pleasant. Note that the search for the specified user(s) -- whether or not "-all" is used -- still assumes that a "login *free -who" login will be used on the appropriate remote Host(s), followed by "who userid". This is why responses to the expanded who command must be so rigidly specified. Note also that regardless of whether the inquiry is made in terms of Network-wide or local user name, the response must be appropriate for use in "concom". Further, if multiple inquiries are made, responses must be in the order received.

To enable console to console communications: "concom -on" (abbreviation: "cc"); to disable, "concom -off". Default is off. To enter message-sending mode: concom username -at hostname ("-at" clause is optional). To exit from message-sending mode, type a line consisting of only a period (cf. Mail, above). The first message sent by concom must be prefaced by an identifying line, beginning "From:" and containing an appropriate address to which to reply. The closing period-only line should be transmitted, so as to allow the other concom to close as well. Acceptable error response is "Not available: username." (which neither confirms nor denies the existence of the particular user -- a matter of concern on the security front). The command must, of course, do whatever is necessary to transmit the messages; i.e., if locally invoked, access the local mechanism, and if invoked for remote communications, access the remote Host's concom command (via "login *free -concom").

"Good" concom implementations will presumably do an expanded who command automatically, so as to spare the user the necessity of having to do it separately. Indeed, the -concom control argument to login is defined to imply the ability to do a who as well as a concom to cater to this possibility. It is tempting to legislate that such an approach be the rule, but the implementation implications are not quite clear enough to do so. The implicit who should be viewed as a strong hint to implementers, though.

File Creation and Manipulation

For file manipulation commands, the user could enter the File Transfer Protocol environment. However, the FTP user interface is constrained by a very high degree of program-drivability. It also lacks abbreviations and suffers from the lack of mnemonicity dictated by limiting command names to four characters. Further, some valuable functions (such as causing a file to be typed out) are not dealt with. Therefore, various UULP file manipulation commands are given in Appendix 1. They need not be addressed in detail here. However, some context would be useful:

The file manipulation commands assume that all Servers have some notion roughly corresponding to "the user's working directory". All file names, whether the yet to be invented Network Virtual Pathname or the "local" variety, are taken to refer to files in this directory unless otherwise indicated. That is, the user should not have to furnish "usr:" or the like; it is taken as given that when he refers to file "x" he means "the file named 'x' in my current working directory" and the Server "knows" what that means.

On the topic of file creation, matters are rather complicated. It is clear that the ability to create files in the UULP environment is extremely desirable. It is also clear that using mail to a fake address to get the file created, then renaming the

"recent mail" file is too byzantine to expect users to do. Unfortunately, it is not clear exactly what the alternative is. That is, it's fairly clear that we need a common editor, but it's not at all clear which editor it should be.

Two widely-known editors come to mind: TECO and QED. However, not everybody has them. Even if everybody did, the "dialects" problem is bound to be a large one. Even if all the relevant system programmers could agree, there remains the question of whether the intended user population would be willing to bother learning a language as complex as TECO or QED. Therefore an optional UULP command to be called "edit" is proposed. This editor is a line-oriented context editor (no "regular expressions", but also no line numbers). It is copiously documented in Chapter 4 of the Multics Programmers' Manual, including an annotated listing of the (PL/I) source code. A simple user's guide may be found in Appendix 3. It's easy to use and cheap to implement. It may also be repugnant to some of the system programmers who would be called upon to implement it -- which is why it is optional, until and unless higher authority makes it mandatory.

Other Protocols

The nominal initial impetus for proposing a UULP was to allow new Network user protocols to be invokable through a common mechanism, rather than requiring a new responding mechanism to be built for a new contract socket for each new protocol. Although this goal has been shunted into the background by the admission of the true goal of the UULP, it has not been dropped completely. Therefore, to enter the FTP Server environment, the UULP command is "ftp"; to enter the RJE Server environment, the UULP command is "rje". Exit is as per the respective protocols.

Other Matters

Programming languages are much too big a problem to tackle here. However, assuming that a user somehow manages to create a source program, he might want some commonality of spelling in invoking the appropriate compiler, or even the object program. As an optional UULP command, then, "call name" should invoke object program name (where the named program may be a "native" command with arguments specified as appropriate). The values "-pli", "-basic", "-fortran", "-lisp", etc., should be recognized as requesting the invocation of the appropriate language processor (to operate on a named source file or interpretively/interactively if no source file was named), with "reasonable" defaults in effect. Note that this all is meant to imply that "native" commands are not directly invokable from the UULP environment (other than by "call"), to avoid potential naming conflicts between system commands and new UULP commands.

On the topic of logging out, the UULP command is "logout" (abbreviation: out). The Server must close the Telnet connection after doing whatever is appropriate to effect a logout. To retain the telnet connection, "logout -save". Having the Server close is viewed as a convenience for the user, in that it spares him the necessity of causing his User Telnet to close. (It is assumed to be no hardship on Servers to step beyond the atavistic position that anybody who logs in to them is at a hardwired local terminal.)

Finally, a comment seems to be in order on error messages. Some explicit messages have been legislated herein. Many areas have not been dealt with, though. It is the spirit of the UULP that error messages should be composed in full awareness of the difference between conciseness and terseness, and should err, if necessary, on the side of over-explaining rather than under-explaining. Another way of putting it is that "CAN'T" is an error message which violates the spirit of the UULP (and the sensibilities of its propounder).

APPENDIX 1. THE COMMON COMMAND SUBSET

Syntax

Abbr

Opt

I. "Set-up" Commands

login id arg in
 The id may be Network-wide or Host-specific.
 "*free" is reserved.
 The arg may be "-mail", "-who", "-concom",
 or may be absent.
 Result is to be either logged in or passed
 off to appropriate daemon.

prompt char or
 Specifies that char is to become or
 preceed the normal prompt message.
 Acceptable prior to login.

erase char er X
 Specifies that char is the erase character.
 Invocation with no argument reverts to default.

kill char X
 Specifies that char is the kill character.
 Invocation with no argument reverts to default.

`eof char` X
Specifies that `char` is the newline character.
Invocation with no argument reverts to default.

`local` loc
Enter the local command environment.

`ftp`
Enter the FTP environment.

`rje`
Enter the RJE environment.

`logout` out
Logout and sever the Telnet connection.

`logout -save` out
Logout but Keep the Telnet connection.

`map`
Apply the case-mapping conventions of Appendix 2.
Required on Hosts to which case is significant.

II. Communications Commands

`readmail` rm
Type out "mailbox".

`readmail (id) -at host` rm X
Type out "mailbox" on remote Host `host`.
Multiple Hosts may be specified,
separated by spaces (blanks).
Implies ability to change working directory
at `host` to directory implied by known
user identifier, or (optionally) by `id`.

`readmail -all` rm XX
Search for mail.
Extremely optional..

`mail id`
Collect input until line consisting of
only a period (".") for mailing to local
user specified by `id`.

`mail -f file id`
Send contents of specified file to specified
local user.

`mail id -at host`
Collect input until line consisting of

only a period (".") for mailing to remote user(s) at specified host(s). Both id and host may be multiple, separated by spaces. (If multiple, they should be taken pairwise.)

mail -f file id -at host
Send contents of specified file to specified remote user(s).

who
The generic who's logged in command.

who id
Is id logged in? Constrained responses.

who id -at host
Is the specified user logged in at the specified host. Constrained responses.

concom -on cc
Enable console to console communications.

concom -off cc
Disable console to console communications.

concom id cc
Send messages to specified local user until line consisting of only a period (".").

concom id -at host cc
Send messages to specified remote user.

III. File Commands

type path print
Type out the contents of the specified file. Pathname may be local or Network-wide. Default to current working directory.

listdir ld
List the contents of the current working directory. (Local format acceptable.)

listdir path ld
List the contents of the specified directory.

rename old new rn change
Change the specified file's name as indicated.

addname old new an x
Give the specified file the specified extra name.

`delete path` ~~remove~~ dl
 Get rid of the specified file.
 ("Expunge" if necessary.)

`copy from to`
 Make a copy of the file specified by the first pathname
 at the second pathname.

`link from to` X
 If your file system has such a concept, make a "link"
 between the two pathnames. If no second argument,
 use same entry name in working directory.

`status path` st X
 If your file system has such a concept, give status
 information about the specified file or directory.

`cwd path` X
 If no argument, return to the "home" directory.

`pwd` X
 Type out the pathname of the current working directory.

`edit path` X
 See Appendix 3.

IV. Invoking "Native" Programs

`call name (args)` ~~run~~ execute
 Invoked the specified program with the
 specified arguments (if any).
 The following names are reserved to indicate the
 invocation of the corresponding language processor:
 "-pl1", "-basic", "-fortran", "-lisp".
 (If no source file indicated, invoke "interpretively"
 if possible.)

V. On-line Documentation

`help name` ?
 Type out information about the specified UULP command.
 If `name` is "-sys", type out information about how to
 use the local system's help mechanism; if
 "-uulp", about the local system's UULP implementation.
 If no name given, describe the command itself.

APPENDIX 2. MAP COMMAND CONVENTIONS

This appendix will eventually contain the case-mapping conventions detailed in RFC 411.

APPENDIX 3. EDIT COMMAND REQUESTS

This appendix will eventually contain descriptions of the edit command requests (to be lifted from my old Multics Primer). For now, it should be sufficient to point out that the requests are basically locate, next, top, change, save, and quit -- i.e., it's the "old-fashioned" flavor of context editor.

Note: Both Appendix 2 and Appendix 3 will be supplied in full when I have the time and strength to do so. Both will be lightly edited versions of existing documents. In neither case are the details particularly important to the overall task of specifying a UULP, although they will of course be important to implementing one.